CLAIMS

1. A method for reducing a usage of main memory by a first class loader and a second class loader, the first class loader and the second class loader being capable of dynamically loading a class having a class file, the first class loader being capable of translating the class file into a first class type and the second class loader being capable of translating the class file into a second class type, the method comprising:

dividing a runtime representation of the first class type into a first loader independent part and a first loader dependent part;

determining whether a runtime representation of the second class type can use the first loader independent part of the runtime representation of the first class type; and

if the first loader independent part of the runtime representation of the first class type can be used by the runtime representation of the second class type,

generating a second loader dependent part of the runtime representation of the second class type using the first loader independent part of the runtime representation of the first class type.

2. A method as recited in claim 1, wherein if the first class type cannot be used by the runtime representation of the second class type, the method further comprises:

generating from the second class file a second loader dependent part of the runtime representation of the second class type and a second loader independent part

3. A method as recited in claim 1, the method further comprising:

determining a satisfaction of a first condition, the first condition being defined by a first class file being the same as a second class file used by the second class loader to define the second class type;

determining the satisfaction of a second condition, the second condition being defined by a loader independent part of the runtime representation of a super class type of the second class type being the same as a loader independent part of a runtime representation of a super class type of the first class type; and

determining the satisfaction of a third condition, the third condition being defined by the second class type having the same unimplemented methods as the first class type,

wherein determining the first condition, the second condition, and the third condition enables the runtime representation of the second class type to use the first loader independent part of the runtime representation of the first class type.

4. A method as recited in claim 1, wherein the class file encodes an architecturally neutral binary representation of the class.

5. A method as recited in the claim 3, wherein the first class file being the same as the second class file used by the second class loader to define the second class type includes,

identifying that each byte of the second class file is the same as a corresponding byte of the first class file.

6. A method as recited in claim 3, wherein the loader independent part of the runtime representation of the super class type of the second class type being the same as the loader independent part of the runtime representation of the super class type of the first class type includes,

identifying a reference to the loader independent part of the super class type of the second class type being the same as a reference to the loader independent part of runtime representation of the super class type of the first class type.

7. A method as recited in claim 6, wherein the reference to the loader independent part of the runtime representation of the super class type of the first class type is stored in the loader dependent part of the runtime representation of the super class type of the first class type, and the reference to the loader independent part of the runtime representation of the second class type is stored in the loader dependent part of the runtime representation of the super class type of the second class type.

8. A method as recited in claim 3, wherein a loader independent part of the runtime representation of the class type includes bytecodes of all the methods of the class.

9. A method as recited in claim 8, wherein the second class type having the same unimplemented methods as the first class type includes,

determining whether the second class type declares an interface;

if the second class type does not declare the interface, an unimplemented

method defined by the second class type is the same as an unimplemented method declared by the first class type; and

if the second class type declares the interface,

if the method declared in the interface has the same name and the same signature in the first class type and the second class type, determining one of the first class type and the second class type including a definition of the method, and the first class file and the second class file not including the definition of the method; and

if the method declared in the interface does not have the same name and the same signature in the first class type and the second class type, identifying the method of the interface declared in the second class type having a definition in the second class type and the first class type.

10. A method as recited in claim 2, wherein generating the second loader dependent part of the runtime representation of the second class type using the loader independent part of the runtime representation of the first class type includes,

obtaining a template of a first loader dependent runtime representation from the first loader independent part of the runtime representation of the first class type;

replicating the template of the first loader dependent runtime representation to produce the second loader dependent runtime representation;

resetting a value of the second loader dependent runtime representation to an initial state; and

setting a reference to the second loader independent part of the runtime

representation in the second loader dependent runtime representation.

11. A method as recited in claim 1, wherein an instance of the first class type includes a reference to the first loader dependent part of the runtime representation of the first class type.

12. A method as recited in claim 11, wherein the first loader dependent part of the runtime representation of the first class type includes:

a reference to the first loader independent part of the runtime representation of the first class type;

a reference to the first loader dependent part of a representation of a super class type of the first class type;

a first loader dependent part table including information computed from symbolic links; and

an array of references to a first loader dependent part runtime representation of methods.

13. A method as recited in claim 12, wherein the first loader dependent part of the runtime representation of the method includes:

a reference to a constant pool cache; and

a reference to a sharable runtime representation of the method.

14. A method as recited in claim 3, wherein the third condition is satisfied if the first condition and the second condition have been satisfied and the first class type

does not declare an interface.

15. A method as recited in claim 3, wherein determining if the first class file is the same as the second class file comprises:

computing a Secure Hash Algorithm-1 ("SHA-1") digest for the first class file and the second class file; and

comparing the SHA-1 digest of the first class file with the SHA-1 digest of the second class file,

wherein the first class file is the same as the second class file if the SHA-1 digest of the first class file and the SHA-1 digest of the second class file have equivalent values.

16. A method as recited in claim 15, wherein the SHA-1 digest of the first class file of the first class type is stored in a table configured to map the first class name of the first class type to a record including the SHA-1 digest of the first class file and a reference to the first loader independent part of the runtime representation of the first class type.

17. A method as recited in claim 16, wherein the first loader independent part of the runtime representation of the first class type includes a pointer to the loader independent part of the runtime representation of the super class type of the first class type.

18. A method as recited in claim 10, wherein the first loader independent part of the runtime representation of the first class type includes a pointer to the first loader

dependent part of the runtime representation of the first class type that is the template.

19. A method for using a first loader independent part of a runtime representation of a software component defined by a first component loader as a second loader independent part of a runtime representation of the software component defined by a second component loader, the method comprising:

determining a satisfaction of a first condition, the first condition being defined by a first binary representation of the first software component being the same as a second binary representation of the second software component used by the second component loader to define a second software component type;

determining the satisfaction of a second condition, the second condition being defined by a second loader independent part of a runtime representation of a super software component type of the second software component type being the same as a loader independent part of a runtime representation of a super software component type of the first software component type; and

determining the satisfaction of a third condition, the third condition being defined by the second software component type having the same unimplemented methods as the first software component type.

20. A method as recited in the claim 19, wherein the first binary representation of the first software component being the same as the second binary representation of the second software component used by the second component loader to define the second software component type includes,

identifying that each byte of the second binary representation of the software

component is the same as a corresponding byte of the first binary representation of the software component.

21. A method as recited in claim 19, wherein the loader independent part of the runtime representation of the super software component type of the second software component type being the same as the loader independent part of the runtime representation of the super software component type of the first software component type includes,

identifying a reference to the loader independent part of the runtime representation of the super software component type being the same as a reference to the loader independent part of the runtime representation of the super class type of the first software component type.

22. A method as recited in claim 21, wherein the reference to the loader independent part of the runtime representation of the super software component type of the first software component type is stored in the loader independent part of the runtime representation of the first software component type and the reference to the loader independent part of the runtime representation of the super software component type of the second software component type is stored in the loader independent part of the runtime representation of the software component type of the second software component type.

23. A method as recited in claim 21, wherein a component loader independent part of the runtime representation of a software component type includes

a bytecode of a method.

24. A method as recited in claim 19, wherein identifying the second software component type having the same unimplemented methods as the first software component type includes,

determining whether the second software component type declares an interface;

if the second software component type does not declare the interface, an unimplemented method defined by the second software component type is the same as an unimplemented method declared by the first software component type; and

if the second software component type declares the interface,

if the method declared in the interface has the same name and the same signature in the first software component type and the second software component type, including one of a definition in the first software component type and the second software component type and not including the definition in the first software component type and the second software component type; and

if the method declared in the interface does not have the same name and the same signature in the first software component type and the second software component type, identifying the method declared in the second software component type having a definition in the second software component type and the first software component type.

25. A computer program embodied on a computer readable medium for

reducing a usage of main memory by a first class loader and a second class loader, the first class loader and the second class loader being capable of dynamically loading a class having a class file, the first class loader being capable of translating the class file into a first class type and the second class loader being capable of translating the class file into a second class type, the computer program comprising:

program instructions for dividing a runtime representation of the first class type into a first loader independent part and a first loader dependent part;

program instructions for determining whether a runtime representation of the second class type can use the first loader independent part of the runtime representation of the first class type; and

if the first loader independent part of the runtime representation of the first class type can be used by the runtime representation of the second class type,

program instructions for generating a second loader dependent part of the runtime representation of the second class type using the first loader independent part of the runtime representation of the first class type.

26. A computer program embodied on a computer readable medium for using a first loader independent part of a runtime representation of a software component defined by a first component loader as a second loader independent part of a runtime representation of the software component defined by a second component loader, the computer program comprising:

program instructions for determining a satisfaction of a first condition, the first condition being defined by a first binary representation of the first software component being the same as a second binary representation of the second software component

used by the second component loader to define a second software component type;

program instructions for determining the satisfaction of a second condition, the second condition being defined by a second loader independent part of a runtime representation of a super software component type of the second software component type being the same as a loader independent runtime representation of a super software component type of the first software component type; and

program instructions for determining the satisfaction of a third condition, the third condition being defined by the second software component type having the same unimplemented methods as the first software component type.